**Exercise 4 : Difference Between JPA, Hibernate, and Spring Data JPA**

**Java Persistence API (JPA) :-**

The Java Persistence API provides a specification for persisting, reading, and managing data from your Java object to relational tables in the database.

JPA stands for Java Persistence API. It is a standard specification (defined in JSR 338) introduced by Oracle to manage relational data in Java applications using object-relational mapping (ORM) techniques.

Think of JPA as a set of guidelines or rules for how to map Java objects to relational database tables.

**Key Characteristics:**

* API Specification Only: JPA doesn't implement anything. It defines interfaces and annotations like @Entity, @Table, @Id, @GeneratedValue, @OneToMany, etc.
* POJO-based: Java objects can be mapped to database tables without needing to extend special classes.
* Annotation-based Configuration: Uses Java annotations to define the mapping.
* Queries via JPQL: JPA uses JPQL (Java Persistence Query Language), which is object-oriented and independent of the underlying database.

**Common Providers (Implementations):**

* Hibernate (most popular)
* EclipseLink
* OpenJPA

**Hibernate :-**

Hibernate is an object-relational mapping solution for Java environments. Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables.

Hibernate provides a reference implementation of the Java Persistence API that makes it a great choice as an ORM tool with the benefits of loose coupling.

**Example**: Below diagram shows an *Object Relational Mapping* between the **Student** Java class and the **students**tablein the database.

**Key Features of Hibernate:**

* Implements JPA: Offers support for JPA annotations and entity management.
* Extended Capabilities: Provides extra features beyond JPA like caching (L1/L2), custom types, native SQL, lazy/eager fetching strategies.
* Session Management: Uses SessionFactory and Session to manage persistence.
* **Transaction Handling**: You handle transactions using Hibernate APIs unless integrated with a framework like Spring.

**How it Works:**

* Maps Java objects (POJOs) to DB tables.
* Uses .hbm.xml files (old) or annotations (modern) for mapping.
* Provides its own query language **HQL** (Hibernate Query Language), though it also supports JPQL.

**Spring Data JPA :-**

Spring Data is a part of the Spring Framework. The goal of Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

Spring Data JPA is not a JPA provider. It is a library/framework that adds an extra layer of abstraction on the top of our JPA provider (like Hibernate).

**Key Benefits:**

* **Reduces boilerplate**: You don't have to write EntityManager or Session code manually.
* **Repository Interfaces**: Just extend JpaRepository, CrudRepository, etc., and Spring auto-generates methods.
* **Query Derivation**: Method names like findByName, findByEmailAndStatus auto-generate the SQL queries.
* **@Transactional Support**: Automatically handles transaction boundaries with Spring's @Transactional annotation.
* **Pagination & Sorting**: Built-in support via Pageable and Sort.

**How It Works:**

And Spring will provide implementations for methods like:

* findAll()
* save()
* deleteById()
* findById()
* And even findByNameContaining(String keyword) if the method name matches JPA naming conventions.

**Code Comparison :-**

1. **Hibernate (Manual Session/Transaction Handling)**

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

1. **Spring Data JPA (Minimal Code)**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}